Занятие 9:
Многопоточность. Часть II

План

I. Серивисы-исполнители (ExecutorService)
   i. Java.util.concurrent.Executors
II. Синхронизаторы
   i. Semaphore
   ii. CountDownLatch
   iii. CyclicBarrier
   iv. Lock и его реализации

ExecutorService

Интерфейс java.util.concurrent.ExecutorService представляет собой механизм асинхронного выполнения, который способен выполнять задачи в фоновом режиме.

- execute(Runnable)
- submit(Runnable)
- submit(Callable)
- invokeAny(Collection<Callable>)
- invokeAll(Collection<Callable>)

Executors

- newFixedThreadPool
- newCachedThreadPool
- newSingleThreadPool
- newScheduledThreadExecutor
- newWorkStealingPool

Fixed thread pool

ThreadPoolExecutor под капотом.

Особенности — создается с фиксированным пулом потоков, без возможности расширения. Если количество запущенных потоков достигло максимума, то новые, добавляемые задачи, будут ожидать освобождения хотя бы одного.

Cached thread pool

ThreadPoolExecutor под капотом.

Особенности — создается с пустым пулом потоков, может расширяться. Удобно использовать, когда необходимо автоматически расширять пул потоков. Обычно применяется при выполнении однотипных «коротких» задач.

Single thread executor

ThreadPoolExecutor под капотом.

Особенности — создается только с одним потоком в пуле. Гарантирует, что задачи будут выполняться последовательно. Если поток «умирает» его место займет другой поток и выполнит следующую задачу.

Scheduled thread pool

`ScheduledThreadPoolExecutor extends ThreadPoolExecutor` `под капотом.`

Особенности — позволяет запускать задачи по расписанию, с различными настройками. Можно запускать задачи периодически или с задержкой.

Work stealing thread pool

ForkJoinPool под капотом.

Особенности — использует все доступные процессоры для достижения максимального параллелизма. Относительно новый механизм, появившийся в Java 7. Оперирует RecursiveTask и RecursiveAction, вместо Callable и Runnable, соответственно. Умеет балансировать нагрузку среди потоков.

Синхронизаторы

Синхронизаторы – вспомогательные утилиты для синхронизации потоков, которые дают возможность разработчику регулировать и/или ограничивать работу потоков и предоставляют более высокий уровень абстракции, чем основные примитивы языка (мониторы).

Semaphore

Реализует шаблон синхронизации Семафор.

Позволяет ограничить количество одновременно выполняемых потоков.

CountDownLatch

CountDownLatch позволяет потоку ожидать завершения операций, выполняющихся в других потоках.

Cyclic barrier

CyclicBarrier выполняет синхронизацию заданного количества потоков в одной точке. Как только заданное количество потоков заблокировалось (вызовами метода await()), с них одновременно снимается блокировка.

Lock

Интерфейс **Lock** из пакета java.util.concurrent – это продвинутый механизм синхронизации потоков. По гибкости он выигрывает в сравнении с блоками синхронизации. Для работы с этим интерфейсом необходимо создать объект одной из его реализаций.

Основные отличия между Lock и синхронизированными блоками:

- Синхронизированные блоки не гарантируют, что сохранится порядок обращения потоков к критической секции;
- Нельзя выйти из синхронизированного блока по времени ожидания (timeout);
- Синхронизированные блоки должны полностью содержаться в одном методе. Lock может быть захвачен в одном методе, а освобожден в другом.

Что почитать

Кей С. Хорстманн, Гари Корнелл. **Java. Библиотека профессионала. Том 1. Основы**;
Стив Макконнелл. **Совершенный код**;
Брюс Эккель. **Философия Java**;
Герберт Шилдт. **Java 8: Полное руководство**;
Герберт Шилдт. **Java 8: Руководство для начинающих**.